NPS52-86-001

NAVAL POSTGRADUATE SCHOOL

Monterey, California

Super Database Computers:
Hardware and Software Solutions for Efficient
Processing of Very Large Databases

David K. Hsiao

January 1986

Prepared for:

Chief of Naval Research
Arlington, VA 22217

NAVAL POSTGRADUATE SCHOOL
Monterey, California

Rear Admiral R. H. Shumaker
Superintendent

D. A. Schrady
Provost

The work reported herein was supported by grants from the Department of Defense STARS Program and from the Office of Naval Research.

This report was prepared by:

David K. Hsiao

DAVID K. HSIAO
Professor
Computer Science

Reviewed by:

VINCENT LUM
Chairman
Department of Computer Science

Released by:

KNEALE T. MARSHALL
Dean of Information and
Policy Science

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER: NPS52-86-001
2. GOVT ACCESSION NO.:
3. RECIPIENT'S CATALOG NUMBER:

4. TITLE (and Subtitle): Super Database Computers: Hardware and Software Solutions for Efficient Processing of Very Large Databases
5. TYPE OF REPORT & PERIOD COVERED:
6. PERFORMING ORG. REPORT NUMBER:

7. AUTHOR(s): David K. HSIAO
8. CONTRACT OR GRANT NUMBER(s): 61153N: RR014-08-01 N0001485WR24046

9. PERFORMING ORGANIZATION NAME AND ADDRESS: Naval Postgraduate School, Monterey, CA 93943-5100
10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS:

11. CONTROLLING OFFICE NAME AND ADDRESS: Chief of Naval Research, Arlington, VA 22217
12. REPORT DATE: January 1985
13. NUMBER OF PAGES: 26

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office):
15. SECURITY CLASS. (of this report): UNCLASSIFIED
15a. DECLASSIFICATION/DOWNGRADING SCHEDULE:

16. DISTRIBUTION STATEMENT (of this Report): Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report):

18. SUPPLEMENTARY NOTES:

19. KEY WORDS (Continue on reverse side if necessary and identify by block number):

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this paper, the notion and characteristics of very large databases for online storage and processing are motivated. The database computer requirements for very large databases are given. The limitations and bottlenecks of the conventional database computer (i.e., the database management system, DBMS, utilizing either the mainframe computer or the backend computer) are delineated. Unlike the database computer for small and simple databases, the database computers for very large and complex databases cannot rely on the upgrade of a conventional mainframe or a backend computer. Nor can it rely on the latest

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

introduction or version of the DBMS. Instead, database computers for very large databases require new hardware organizations and novel software techques in order to handle the databases cost-effectively and performance-efficiently.

This paper recommends the kinds of hardware architectures and software techniques which may make database computers for very large and complex databases effective in both operation and cost and efficient in both response time and transaction throughput.

SUPER DATABASE COMPUTERS: HARDWARE AND SOFTWARE SOLUTIONS FOR EFFICIENT PROCESSING OF VERY LARGE DATABASES*

David K. Hsiao

Department of Computer Science
Naval Postgraduate School
Monterey, CA 93943
U.S.A.

Abstract

In this paper, the notion and characteristics of very large databases for online storage and processing are motivated. The database computer requirements for very large databases are given. The limitations and bottlenecks of the conventional database computer (i.e., the database management system, DBMS, utilizing either the mainframe computer or the backend computer) are delineated. Unlike the database computer for small and simple databases, the database computers for very large and complex databases cannot rely on the upgrade of a conventional mainframe or a backend computer. Nor can it rely on the latest introduction or version of the DBMS. Instead, database computers for very large databases require new hardware organizations and novel software techniques in order to handle the databases cost-effectively and performance-efficiently.

This paper recommends the kinds of hardware architectures and software techniques which may make database computers for very large and complex databases effective in both operation and cost and efficient in both response time and transaction throughput.

(This paper is to appear in the
Proceedings of IFIP Congress '86)

* The work reported herein is supported by grants from the Department of Defense STARS Program and from the Office of Naval Research and conducted at the Laboratory for Database Systems Research, Naval Postgraduate School, Monterey, CA 93943.

1. INTRODUCTION

The notion and characteristics of very large databases are motivated and presented in this section. In particular, we ask the question: what is a very large database? A very large database can be characterized by its size, growth, complexity, performance and cost. To this end, we are introducing each of the characteristics in the following subsections. We are also including in the subsections the database computer requirements for handling the unique characteristics of the very large databases.

We will not motivate the need of very large databases. As we are entering the information age and high-tech era, where we are increasingly relying on the use of database-oriented and database-driven systems, the need for very large databases should be obvious.

1.1. The Size of Very Large Databases

In considering the size of a very large database that can be placed in online stores and processed by the computer, we exclude certain kind of storage devices such as magnetic tapes, optical disks, magnetic cartridges, and floppy disks. The exclusion is based on our observation that they have one or more of the following limitations: the sequential-access operation, read-only operation, very slow access time, and very limited capacity. We are primarily interested in high-capacity magnetic disks with both the read and write operations, such as the IBM 3380 disk drive which has slightly over one gigabyte (10^9 bytes) of storages capacity [1]. Consequently, today a manageable and processable database of very large sizes is in tens of gigabytes.

The aggregation of many locally-processed databases interconnected by a computer network may be in hundreds of gigabytes. However, if we cannot process a local database efficiently and effectively, we cannot expect the network to provide effective and efficient database operations. Thus, we will not focus on the aggregate size of the locally-processed databases. Instead, we focus on the individual sizes of the locally-processed databases. Such a database is limited by the current capacity of the database store to tens of gigabytes (say, about 10^{10} bytes).

The storage density of magnetic disks tends to double every other year; we can expect the trend to continue in the next decade. Also, with the maturity of magneto-optical disks, the introduction of vertical recording, and the use of thin-film read/write heads, we may in ten years have very large databases in hundreds of gigabytes (say, reaching a terabyte, i.e., 10^{12} bytes in size) [2]. Thus, when we characterize the sizes of very large databases for the present and for the near future, we are referring to sizes of a few gigabytes to several gigabytes on the basis of our understanding of the disk technology and trend. The database computer for very large databases is therefore required to handle tens and hundreds of gigabyte disks.

1.2. The Growth of Very Large Databases

Most databases tend to grow in size. As the storage density increases and the cost per byte for storage decreases, there is the incentive to have even larger databases.

Updated data may serve as the "corporate memory" of the past and present. Consequently, we tend to accumulate data instead of purging data. In one decade we may grow from a very large database of few gigabytes into an even larger database of several gigabytes. Thus, a very large database is characterized by its tendency to grow larger.

One of the requirements of very large databases is that despite their growth the response times of the transactions written for the databases must stay invariant. In other words, we do not want the response time of a transaction changing, say, from several seconds to a few minutes over a period of database growth. Whereas seconds may be acceptable for now, minutes may not be acceptable for the near future.

As the databases are growing in size, there may be an increase in the number of users of the databases. Consequently, both the frequency of the usage and the number of transaction will increase which, in turn, may increase the response times for the existing and new transaction. Any such increase in response times will render the use of the very large and growing database time-consuming. By maintaining and even improving the response times of the transactions against the very large and ever growing databases, the database computer can continue to make the databases useful and viable to the user during the database growth period. Both the response-time invariance in spite of the database growth and the response-time improvement for all transactions for stable databases are the requirement for a high-performance database computer.

1.3. The Complexity of Very Large Databases

Unlike physical resources of a computer where one piece of a physical resource (say, a reel of blank tape) does not have information related to another piece of the physical resource (say, another reel of blank tape), data in a database are used to represent related information. In fact, most of these relationships are also represented in data. This is because we are not merely using the data as information items (i.e., physical resources) so that we must schedule and manage their utilization, but we are also using the data for related information (i.e., logical resources) so that we can process the related data for our transactions and manipulate the relationships for deriving new information and relationships.

Data models have been used to represent the relationships. In a contemporary database management system (DBMS) where the database is small and simple, the DBMS supports only a single data model and model-based data language. Consequently, we have, for example, three separate types of DBMS, one is the relational DBMS which supports the relational data model and relational data language, one is the hierarchical DBMS which supports the hierarchical data model and hierarchical data language and one is the Codasyl DBMS which supports the Codasyl data model and Codasyl data language. For different applications, we may thus use, for example, a relational DBMS for handling tables, forms, and ad hoc queries a hierarchical DBMS for managing designs of assemblies, subassemblies, components and parts, and a Codasyl DBMS for exercising inventory control of supplies and demands [3,4].

For a very large and complex database where the database applications are diverse, and involved, there may be applications for example, in table handling, design management and inventory control. A single data model and model-based data language will

not suffice, since what the model and language are good for in one application may not be adequate for in another application. What we need for very large databases is a single database computer which can support a variety of data models and a large number of model-based data languages. With the presence of multiple models and languages, the database computer allows the user to explore the strong points of these models and languages for their applications. Consequently, stored data and written transactions may best be developed for the intended applications. The requirement for multiple data models and data languages in a single database computer is prompted by the characterization of the very large and complex database of diverse and involved applications.

1.4. The Performance of Very Large Databases

Regardless of the models being used to characterize the databases, there are two kinds of stored data -- the meta data and the base data. The meta data consist of the relationships of and facts about the base data. The base data are the literal representation of the information units. A database is therefore a collection of meta data and their base data. The meta data are different from the base data in size, kind and operation. If the database is small and simple, we may overlook their differences and store and process them together. However, for very large databases, these differences are pronounced and have great impact on performance. They must therefore be stored, processed and utilized differently. In size, the meta data are much smaller than the base data in a database, say, ten per cent of the database.

In kind, the meta data consist of some schemas and descriptors and mostly indices of the base data whereas the base data consist of attribute values. Due to their difference in size and kinds, a page of meta data may comprise, for example, many indices whereas a page of base data may comprise a few sets of attribute values. Thus, we use blocks to hold records, i.e., full sets of attribute values. Consequently, the storage structures of the meta data and of the base data are different.

For performance reasons, we may desire to access a large number of smaller pages of meta data quickly, and to access a larger number of larger blocks of base data readily. The performance issue is particularly acute for a very large database where the meta data may be in tens of gigabytes and the base data in hundreds of gigabytes. What we want to do is to partition the meta data in terms of their attributes so that the required indices in the desired partitions can be found with very few accesses to the disks and without involving the majority of the other partitions. Furthermore, disk accesses for different pages of a partition should be made simultaneously to allow parallel processing of the partition. This requirement suggests that for a database computer, the meta data of the database are to be processed in a page-(index-)serial-and-partition-parallel fashion, so that a large number of smaller pages of meta data can be accessed quickly. For the base data of the database, the attribute-value sets (conventionally called records or tuples) may have to be clustered on the basis of both explicit and implicit indices. In this way, high performance is achieved by narrowing on a few clusters for disk accesses. Again, the requirement suggests that for a database computer, the base data of the database are to be accessed in a block-(record-)serial-and-cluster-parallel fashion, so that a large numbers of larger blocks of base data can be accessed quickly.

In the index-serial-and-partition-parallel access operation, the individual index pages are accessed directly and processed serially, i.e., finding an index page directly on the disk and reading the page one index at a time for processing. However, the entire partition of indices having the common attribute or attribute-value range can be processed in parallel. For a partition with hundreds and thousands of indices of the same attribute or attribute-value range, the effective access and processing time of the partition is reduced to the time required to access and process a few (say, one or two) index pages. Thus, a voluminous partition may be accessed and processed at the page rate -- an indication of high-performance meta data management and processing. Similarly, the record-serial-and-cluster-parallel access operation accesses individual record blocks of a cluster directly and processes each accessed block one record at a time; but the blocks of the cluster are accessed and processed in parallel. Consequently, for a large cluster of hundreds or even thousands of records, the effective access and processing time of the cluster is reduced to the time required to access and process a few (say, one or two) record blocks. Thus, large clusters may be accessed and processed at the block rate -- an indication of high-performance base data management and processing.

Typically, for a given transaction, the index-serial-and-partition-parallel operations are initiated first and followed by the record-serial-and-cluster parallel operations. Thus, for a given transaction, there is a set of sequential operations consisting of two sets of parallel operations, namely, the set of index-partition operations and the set of record-cluster operations. Again for the performance reason, it is required that the set of index-partition operations for one transaction is to be overlapped with the set of record-cluster operations for another transaction. In this way, we achieve concurrent operation of two sets of parallel operations without being hindered by their built-in requirements for sequentiality.

The performance of very large database is therefore characterized by the database computer's capability in performing the index-serial-and-partition-parallel and record-serial-and-cluster-parallel operations, the index partition operation at the page rate and the record-cluster operation at the block rates, and the index-partition operation and the record-cluster operation, concurrently.

1.5. The Cost of the Very Large Database

There are three types of cost associated with a very large database. First there is the storage cost of the database. This cost is high despite the low cost per byte of the magnetic disks, since we are dealing with the database in gigabytes and insisting on treating meta data and base data differently. We may have to bear the cost of one smaller set of disks for the meta data and one larger set of disks for the base data. Nevertheless, such expenses are necessary.

What we have to watch out for is the 'fat-disk' phenomenon. A fat disk has many unused disk spaces. In other words, the database loading factor of the disk is low. For example, using hashing schemes to generate record addresses for placing records on the disks for storage is a sure way of creating fat disks. For a small database, a small percentage of wasteful or unused disk space is not a detrimental factor. For a very large database, even a small percentage of unused disk space may mean hundreds of

megabytes or even few gigabytes. Furthermore, it requires the database computer to support more disk drives (since they are fat) which complicates the management and control of the disk systems.

The second cost associated with the very large database is the cost of the database computer. Regardless of the necessary new hardware organizations and software techniques for very large databases, both the hardware and software must be cost-effective. For example, the associative registers and arrays are very useful for predicate-processing and content-addressing. However, associative registers and arrays of several megabytes in size would be prohibitively expensive and impractical.

The third cost associated with the very large database is the cost of upgrade. Since the response times of transactions of a very large, growing database should be invariant, or the response times of transactions of a very large, stable database may need improvement, there is the necessity of upgrading the hardware and software regularly in order to maintain or improve the response-time performance. The cost of regular upgrades must be effective. Ideally, the cost of such upgrades is proportional either directly to the amount of database capacity growth for the same response times or inversely to the amount of response time reductions for the same database capacity.

1.6. A Summary of the Notion and Characteristics of Very Large Databases

From the viewpoint of computer architects and system designers, a computerized, very large database is characterized by its size, growth, complexity, performance and cost. Sizes are in bytes. In this and coming decades, a very large database many have tens of gigabytes of meta data and hundreds of gigabytes of base data. The total size of a very large database may grow to a terabyte. The architects and designers must find ways to handle hundreds of high-capacity disks, e.g., the gigabyte disks.

Since growth is indicative of a very large database, the computer architects and system designers must come up with ways to hold constant the response times of transactions, for the growing database and to improve the response times of the transactions for the stable databases. Response-time invariance for the growing databases and response-time reduction for the stable databases are the very large database issues which must be overcome.

Very large databases are also very diverse in applications, simply because there are more users, more information needs and more transaction types. The complexity of very large databases is measured in terms of the database computer's capability to support multiple data models and multiple model-based data languages. It is these data models which capture various data representations for the information contents. It is these model-based data languages which enable various transactions to be written for the information needs. It is these database applications which provides the diversity and complexity of the very large databases. Multi-model and multi-lingual are two user-system interfaces that the architects and designers must strive for the very large databases.

Database operations for very large databases must be considered in terms of parallel operations on meta data, parallel operations on base data and concurrent execution of meta and base data operations. The meta data operations are index-serial-and-

partition-parallel and the base data operation are record-serial-and-cluster-parallel. In addition, the index-partition operations for one transaction should be concurrently executed with the record-cluster operations for another transaction.

Database operations for very large databases must also be considered in terms of units and rates of data access, transfer and processing. For meta data, the unit should be in page size. The rate of page processing should match the rate of page transfer. For base data, the unit should be in block size, e.g., a disk track size. The rate of block processing should match the rate of block transfer. With matching rates of transfer and processing and with parallel and concurrent operations, the performance of the very large databases may fare better. The architects and designers must provide the necessary hardware and software designs for such performance operations and rates.

Although the system cost of a very large database may be high, the unit cost of database storage and processing should be lower. In fact, it should be lower than the unit cost of database storage and processing for small and simple databases. This is a real challenge to the architects and designers. In other words, how can the architects and designers meet all the other requirements of very large databases such as size, growth, complexity, performance and upgrade with low cost?

In a later section, we will present some low-cost solutions in hardware and software for very large databases. Before presenting the solutions, however, let us look into the issues and problems of the conventional DBMS and the typical database computer in meeting the challenge of the very large database applications.

2. Conventional Solutions to Very Large Databases

In this section, we examine the conventional solutions to database management and processing. In the examination, we attempt to relate the solutions to the characteristics of the very large database outlined in the previous section. More specifically, we delineate the adequacy and inadequacy of the conventional solutions with respect to the issues of size, growth, complexity, performance and cost.

From an architectural viewpoints, there are essentially two conventional classes of solutions: the mainframe-based solutions and the single-backend solutions. We address each of the two classes in the following subsections.

2.1. The Mainframe-Based Solutions to Database Management and Processing

The mainframe-based solutions to database management and processing dominate the field of the conventional database management system (DBMS) where the system software runs as an application of the operating system on the mainframe computer. As depicted in Figure 1, there are several, different applications. For illustrative purposes, we have only shown two types, one for databases and the other for compiler-languages.

Typically, the user writes the transactions in a data language and submits the transactions to the mainframe computer for execution. The operating system of the mainframe computer first causes the DBMS software to be executed, and then passes the

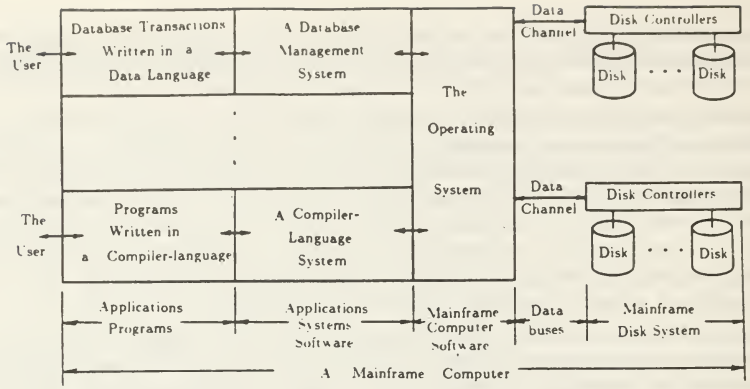

Figure 1: The Mainframe-Based Approach To Database Management and Processing

transactions to the running DBMS. For accessing the database, the DBMS relies on the disk I/O operations and the I/O control subsystem (IOCS) of the operating system. Meanwhile, the database of the DBMS co-exists, for example, with the program files and subroutine libraries of other non-database applications on the disk system of the mainframe computer. It is important to note that the database stored on the disks are modeled on a give data model on which the data language of the transactions is also based.

Similarly, the user may write programs in a given compiler-language, and submit the programs to the mainframe computer for execution. The operating system of the mainframe computer first causes the compiler-language system (CLS) software to be executed, and then passes the programs to the running CLS. For accessing program files and subroutine libraries, the CLS relies on the diskv I/O operations and IOCS of the operating system. Meanwhile, the program files and subroutine libraries of the CLS co-exists with the database of the DBMS on the disk system of the mainframe computer. We note that in this setting the DBMS must share the use and control of the physical resources with all of the other (non-database) applications (such as compiler-language programs) of the mainframe computer.

By sharing the use and control of the physical resources such as data channels, disk controllers and disk drives, the DBMS cannot support very large sizes of databases effectively and efficiently. The ineffectiveness is due to the limitation in storage capacity where the disk space is also being used for other applications. The inefficiency is due to the reliance on a general-purpose and all-embracing operating system to provide disk I/O operations and control. A specialized database operating system for more efficient disk I/O operations and control and for highly parallel access operations of indices and records has not been found in the contemporary mainframe computer.

In database growth, there is no way for the DBMS to hold the response times of the transactions constant. When the database capacity has doubled or tripled, the response times of the same transactions tend to double or triple also. Fine-tuning the DBMS software may only allow a small fraction of the response-time improvement. It does not reduce the response times by one half or two thirds. Replacement of the present mainframe computer with an advanced model may improve the raw performance of the computer hardware such as faster CPU cycles, shorter memory cycles, higher channel capacities, and greater disk-transfer rates. Raw performance gains are measured in fractions (say, 50% faster) and factors (say, 2 times higher). The raw performance gains do not impact the over-all system performance such as the response times proportionally. In other words, the response-time improvements are not in the same corresponding fractions and factors of the raw performance gains. In fact, the corresponding gains tend to disappear at each level rather rapidly, as we measure them at the hardware level, then at the operating-system level, next at the DBMS level, and finally at the transaction level.

Additionally, mainframe replacement is costly. It is costly because we are replacing either the entire computer or the most expensive parts of the computer. New models and subsystems tend to be more expensive at their introduction. The more economical way to improve the system performance is to add and use the same types of hardware. However, as long as we are insisting on the mainframe-based approach, we cannot simply double or triple the number of CPUs, channels, main memories, and controllers in order to hold the response times of the transactions constant for the two-fold or three-fold increase in database capacity. The mainframe-computer components are tightly coupled. They do not lend themselves for hardware extension and software replication.

Mainframe replacement is also disruptive. The upgrade due to database growth (and for response-time invariance also) causes the other non-database applications software to be upgraded to the new replacement. Ideally, the upgrade should be done in real time without any interruption of the on-going programs and transactions. However, the mainframe upgrade (say, replacing with a faster CPU) requires the disruption of the on-going work. While the upgrade for the DBMS is taking place, all other application software are also disrupted.

In performance, there is little room to carry out parallel database operations in a mainframe computer. Even with the clever use of multiple buffers, buffer-switching techniques, multiple channels, and multiple controllers, the degree of channel overlapping of disk I/O operations for the same transaction does not exist. For a major installation, the mainframe may have several channels. However, few of the channels are dedicated to the disk I/O operations of a single transactions. In fact, it is likely that a single channel is shared by the disk I/O operations of several transactions. The shared channel is known as the multiplexer channel. Most channels of a mainframe computer are multiplexer channels.

On the other hand, for the performance of very large database, we desire such operations as index-serial-and-partition-parallel and record-serial-and-cluster-parallel on a per-transaction basis. Consequently, we need tens and hundreds of dedicated channels (known as the selector channels) for simultaneous accesses and transfers of index pages

and record blocks. Such a large number of parallel operations are difficult, if not impossible, to carry out in the mainframe computer. The difficulty lies in the configuration and cost of tens and hundreds of selector channels, the introduction of a database operating system for scheduling and controlling database disk I/O, and the load and presence of other application system software.

Finally, every mainframe-based DBMS provides only a single data model for the database construction and single model-based data language for the transaction development. Thus, if we are interested in the relational model and its SQL data language [5, 6], and the hierarchical model and its DL/1 data language [7, 8], we must have two DBMSes on the mainframe, namely, for example, SQL/DS [9] and IMS [10]. If we are interested in several data models and model-based data languages, we must have several DBMSes. Having multiple DBMS is not only costly, but the databases of the DBMSes, although they reside in the same disk system, are not compatible. Consequently, a database constructed in one data model cannot be accessed by a transaction written in a data language based on another data model.

In summary, the mainframe-based DBMS is adequate for small, simple and stable databases. It cannot support very large databases adequately due (a) to its inability to accommodate very large sizes, rapid growth, and complex applications, (b) to deliver desired performance with or without hardware upgrades and (c) to provide low cost of upgrade, high level of diverse applications, and low level of disruption during upgrades.

2.2. The Single-Backend Solutions to Database Management and Processing

Unlike the mainframe-based approach, the single-backend solutions to database management and processing are a rather recent introduction [11]. To overcome the problems of performance degradation and resource sharing and control, the database-system software is off-loaded from the mainframe computer to a separate, dedicated computer with its own disk system, known as the backend of the mainframe as depicted in Figure 2.

This approach, called single-backend approach, is characterized by the architectural configuration where the DBMS is placed in a dedicated backend computer with its own operating system, disk controllers and disk drivers. As far as the mainframe computer is concerned the presence of the backend computer appears to it as a peripheral system much like the communications frontend computer which handles terminals and serves as the gate-way to a computer network. In comparison with other peripheral systems of the mainframe computers such as the disk system, the tape system and the unit-record devices, the database backend computer consists of considerable software, firmware and hardware. This is because, in addition to the DBMS software, there is the need of interfacing software for the backend and mainframe computers.

As a dedicated computer for the database backend, the software, firmware and hardware of the computer can also be specialized and tailored. For example, larger and faster buffer memories may be used as "disk caches." A faster CPU and IOCS may be included so that the database processing can keep up the rate of disk transfers of the database. Because the physical resources of the backend computer are exclusive to the

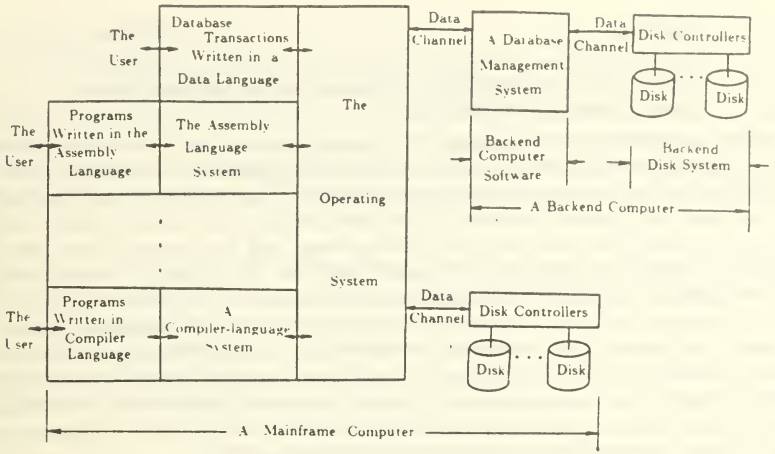

Figure 2: The Single-Backend Approach To Database Management and Processing

backend computer, both the mainframe and backend computers have simpler control and use of their respective resources. A notable absence is the large and software-laden DBMS in the mainframe computer. (See Figure 2 again.) The presence of small interface software in the operating system of the mainframe computer for routing the transactions to the backend and for directing the results to the user has little impact on the mainframe. The major benefit to the mainframe is the absence of the DBMS software which frees up considerable CPU and memory cycles and the main memory space of the mainframe. In fact, the mainframe computer configuration depicted in Figure 2 should provide better overall services to the user than the mainframe computer configuration depicted in Figure 1. Consequently, we may retain the mainframe computer longer and improve its overall performance by adding a single-backend computer for database management and processing cost-effectively.

However, there are limitations of the single-backend approach as was originated at the Bell Laboratories in their work on XDMS [12]. As quoted below, the main goals of XDMS were to:

> (1) obtain a cost saving and a performance gain through specialization of the database operations on a dedicated backend processor,
>
> (2) allow the use of shared databases [by different mainframe computers, now called hosts],
>
> (3) provided centralized [i.e, physical] protection of the

database, and

(4) reduce the complexity when developing software for a stand-alone and new machine.

Single-backend database computers can achieve goals 3, and 4, but have had difficulty in meeting goals 1 and 2 entirely. Although single-backend computers may be cost-effective, these computers suffer from performance problems; in fact they suffer from the same performance problems of the DBMS running on the mainframes. As the use of a single-backend database computer increases and the growth of the database intensifies, the single backend can no longer maintain the desired performance which had been gained by offloading the database software from the mainframe and by utilizing the dedicated software, firmware and hardware. Like the hardware upgrade of mainframe computers, the conventional approach to the hardware upgrade of single-backend database computers is to use the next more powerful backend. Although this type of upgrade is not as disruptive to the mainframes, the upgrade does not yield proportional performance gains in terms of response-time reduction and invariance with respect to very large databases and very rapid database growth.

The single-backend computer is not amenable to parallel operations, since the number of database backends is one. Consequently, high-performance single backends are not in sight. All of the single backend computers support only a single model and its model-based language. The use of multiple single-backend computers, each of which supports a different data model and data language, preclude the sharing of databases as remarked in goal 2. Furthermore, upgrading multiple single-backend computers is not a cost-effective and time-saving effort.

In summary, the single-backend computers are good for small, simple and stable databases. They differ from the mainframe-based DBMS in that they allow more cost-effective upgrade and little disruption to other non-database applications. They also provide better physical protection of the databases and more incentives for retaining the use of the mainframe computers (i.e., hosts) for a longer period of time. Nevertheless, the problems and issues have confronted the mainframe computers on very large database sizes, growth, complexity, performance and cost have not been resolved; they are, instead, merely being deferred to the single-backend computers.

3. Towards Efficient and Effective Management and Processing of Very Large Databases

There is not much of a prospect in improving either the mainframe-based or the single-backend database computer for handling very large databases. The characteristics and requirements of very large databases have overwhelmed the capacity and performance of the conventional database computers and their solutions to database management and processing. What we need are unconventional and innovative solutions to the management and processing of very large databases. These solutions must meet the characteristics and requirements of very large databases and yield substantial

efficiency and effectiveness.

In the following subsections, we recommend an unconventional architecture for very large database management and processing. We discuss the architecture in terms of the characteristics and requirements of the very large databases. In the discussion, we examine the architectural designs in meeting the characteristics and requirements. These designs are the designs of hardware components and software algorithms for very large database operations. We then present these components and algorithms in some detail.

3.1. The Architecture of A Database Computer for Very Large Databases

The new kind of the database computers (depicted in Figure 3) is of the multiple-backend approach where no database system is mainframe-based and each database system consists of one or more backend controllers (starting with one) and one or more backends (beginning with two usually) with their disk systems interconnected by a communications network. Examples of the multiple-backend approach to database management can be found in the experimental Multi-Backend Database System (MBDS) utilizing an Ethernet interconnection [13] and the commercial Teradata DBC/1012 system utilizing a communications-and-sorting network, known as the Y-net [14].

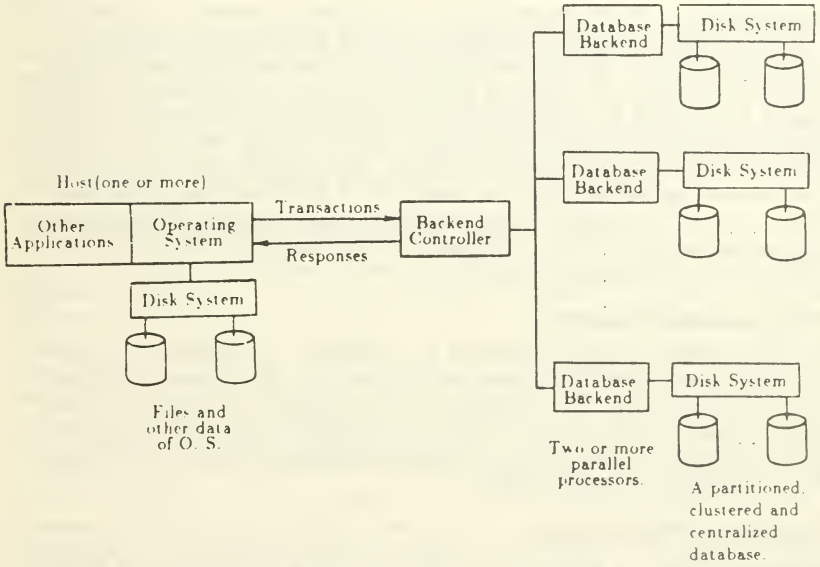

Figure 3. The Multiple-Backend Approach to Very Large Database Management and Processing

Unlike the mainframe-based and single-backend approaches, the multiple-backend approach to database management and processing has resulted in a parallel architecture of database processors and their database stores. Unlike a fixed parallel system, where the number of processors once built-in cannot be changed, the processor-store pair of the multiple-backend database computer can be added and deleted without requiring any reprogramming of the system software or any modification of the system hardware. Thus, this is a variably parallel system. The number of backends (i. e., parallel database processors and their disk systems in a given system) may be in tens as in the case of MBDS or hundreds as in the case of DBC/1012.

3.1.1. The Backend Controllers

All of the backend controllers have identical hardware and replicated software which handle the preprocessing of the transactions, broadcast the transactions to the backends, keep track of the execution progress of the transactions, assemble the responses from the backends, and route the responses to the users or user transactions originated at the host or terminals. The number of backend controllers in a given system is usually one but may be more for redundancy and reliability.

3.1.2. The Interconnecting Network

The interconnecting network can range from a broadcasting network to a cross-bar network. However, since database management involves aggregate functions such as maximum and minimum and sort-and-merge functions such as sequencing and merging (relation joins), the network may have local memories and processors for such functions. Since backends are intended to perform most of the database operations on their meta-data partitions and base-data clusters independent of one another, there are minimal communications among the backends and between the controller and its backends. Thus the interconnecting network does not have to be a high-bandwidth communications network. Instead, the network may assist the backends in performing aggregate and sort-and-merge functions. As there is usually only one controller, the use of broadcasting and tree-like networks becomes common.

3.1.3. The Expandable Capability

The multiple-backend database computer is expandable. The expansion requires only the use of the same backend hardware and the replication of the existing backend software on the new hardware. The redistribution of the partitions and clusters on the old and disks achieves the desirable effect where multiple transactions being executed in the backends are reading (or writing) and processing multiple data streams of partitions and clusters coming from (or going to) disks.

3.1.4. The Database Organization

A database must have its meta data partitioned and its base data clustered at the database-creation time. Each partition (cluster) must be placed on the respective disks of the separate backends one page (block) at a time. The data placement algorithm is usually a round-robin algorithm which attempts to achieve even distribution of the index pages (record blocks) among the index (record) disks of the separate backends. In

the following, we show how this database computer is the likely candidate to provide efficient and effective solutions to the management and processing of very large databases.

3.2. On Handling the Very Large Database Size

Since primary database operations are replicated in all the database backends and each backend can handle a separate disk system, the management and control of a large number of disk systems become an easier task. As far as the disks are concerned, we are applying distributed management, control and processing. Consequently, in a multiple-backend database computer, the task of handling hundreds of disk drives is a well-distributed and straightforward function of the computer.

Due to the use of partitions and clusters and the absence of any replication of base data, the database backend of one disk system is not required to consult the database backend of another disk system for transaction processing and data accesses. Consequently, there is little traffic among the backends regarding their disk systems. Disk access operations, whether they are for the meta data or for the base data, can be carried out in parallel among the backends for a transaction and concurrently among the backends for different transactions without being tightly coupled for synchronization and control.

To increase the size of the database due to insertion and update, the multiple backend database computer facilitates the addition of disk drives to the backends or the expansion of backends and their disk systems. Since large numbers of backends and their disks are intrinsic to the multiple-backend database computer, there is no need to freeze the numbers and to replace the present computer and its disks with the next greater-performance, higher-capacity and more-expensive model. This orderly expansion and addition will also eliminate the disruptive and costly upgrade.

3.3. On Dealing With the Very Large Database Growth

The growth of a database is also characterized by the amounts of the responses to the same transaction. As the database grows, the responses to the same transaction also increases. Consequently, the capacity growth of one multiple-backend computer over the other one is also characterized by the amounts of their responses and the sizes of their databases. What we want to achieve is the response time of a transaction to be held constant despite the capacity growth. To compensate for the extra work necessary in capacity growth, the multiple-backend database computer offers the configuration with additional backends. Obviously, if we are to have, for example, twice the amount of the responses to a transaction in the new configuration over the amount of responses to the transaction in the old configuration, the size of the database in the new configuration is likely to be a multiple of (say, twice) the size of the database in the old configuration. To compensate for the increase in the database size and the response-set size, the new configuration is given a corresponding increase in number of backends and their disk systems. For this example, if the database size disks in the new configuration would be doubled. Therefore, the invariance of the response time of the same transaction is achieved as the size of response sets and the number of backends increase in the same proportions. We characterize the database growth in terms of the response-time

invariance and the present the formula in Figure 4 below.

$$\text{The Response-Time Invariance} = \left(\frac{\text{The Response Time of Configuration } Z}{\text{The Response Time of Configuration } X} \right) - 1$$

Figure 4. The Response-Time-Invariance Formula

Let x , the number of backends for configuration X, be 20, and z the number of backends for configuration Z, be 60, then ideally we would like the ratio of response times of configuration Z of 60 backends and configuration X of 20 backends to be 1. Consequently, the response-time invariance of the 60-backend database computer over the 2-backend database computer would be 0, i.e., no variance. Of course, in this example, the transaction receives three times more responses in the 60-backend database computer than the response to the same transaction in the 20-backend database computer.

This example illustrates that if we triple the number of backends of an existing multiple-backend database computer for the grown database, we would expect to maintain the same response time of the transaction despite the fact that the transaction is now processing three times more responses than before. For the purpose of maintaining the same response time of a transaction, it is ideal if the number of backends added to the existing configuration is in proportion to the increase in the amount of the responses. In this example, we need to add three times as many new backends and their disk systems to the existing database computer in order to hold the response time constant.

In reality, some variance in response times will always exist. The issue is therefore how close a given multiple-backend database computer can maintain its ideal response-time invariances. As shown in Figure 5, we must measure (benchmark) a sufficient number of configurations for a given transaction and a similar number of databases in order to determine the system overheads and their impact on the response times. Preliminary benchmarking results of MBDS are excellent for a small number of backends.

Ideally, we would expect that in Figure 5, $S_i = i$ for $i = 1, 2, ..., n$ and for large n. However. in typical cases, $\delta_1 = 0$ and $\delta_2 < \delta_3 < \cdots < \delta_n$, where δ_i is the system overhead incurred in handling the transaction in the i-backend configuration. In studying Figure 5. we may expect the benchmarking effort to address the following issues:

(1) What are the values of δ_i for the given i-backend computers under benchmarking?

(2) How large will n be when there is no invariance in response time (i.e., $S_n \geq S_{n+1}$)?

(3) How large will n be when the system overhead becomes pronounced (i.e., $\frac{\delta_{n+1}}{n+1} \gg \frac{\delta_n}{n}$)?

Although this new capability of the multiple-backend database system requires additional benchmarking work to determine its overhead (i.e., δ_i) and its threshold

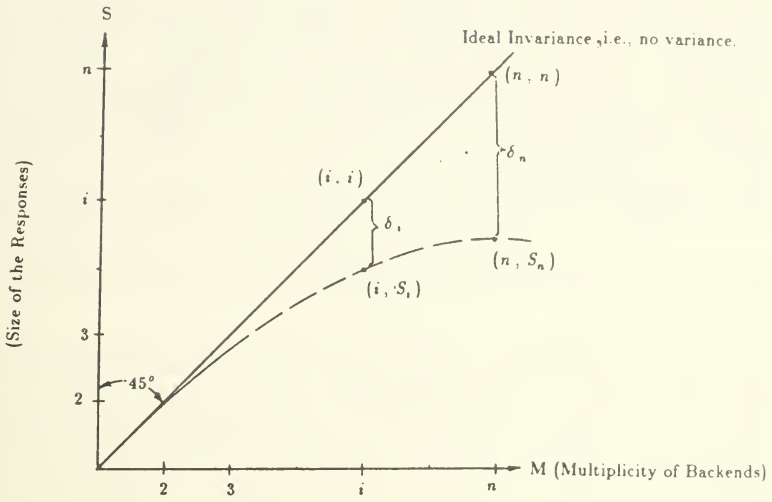

Figure 5. The Response-Time Invariance Measure

(i.e., n where $S_n >= S_{n+1}$), it is promising. In fact, this is the only way to hold the response time constant despite the growth of database and the increase of responses to the transactions. Furthermore, the ease of adding the backend hardware and replicating the backend software without resorting to the next model of the hardware and latest version of the software makes this capability cost-effective and performance-effective. Even if the overhead turns out to be higher at a large threshold, this capability is attractive and viable [15].

3.4. On Resolving the Very Large Database Complexity

The way to resolve the complex and diverse applications of the very large databases is to provide the multi-model and multi-lingual capability via the multiple-backend database computer. Although it is possible to have a large number of backends each of which supports a single and different model and its model-based data language, we forgo this approach. Had we pursued this approach, the backend software would not be identical and replicatable. Furthermore, the multiple-backend database computer would become a collection of heterogeneous DBMSes whose databases and resources could not be shared and optimized for performance gains, capacity growth and system upgrade. Before presenting a new approach to multi-model and multi-lingual capability, let us review the evolution of operating systems.

The early operating systems, like the present database systems, individually supported a specific set of data structures and a single programming language which defines and manipulates the structured data. For example, the Fortran Monitor System of the late fifties supported an operating system environment for a single programming language (i.e., Fortran) and its corresponding data structures (e.g., Fortran arrays and variables). As operating systems evolved through the sixties and seventies and into the eighties, the same operating environment supported a variety of data structures and their programming languages. For example, the Unix operating system supports traditional programming languages, such as C, Pascal, and Fortran, list-processing programming languages, such as Lisp, and logic programming languages, such as Prolog. Each of these programming languages has its own set of data structures. All programs written in the aforementioned languages and data structures can be run in the same operating system which is also responsible for managing all of the physical resources shared by the running programs and their data structures.

Given this characterization of the operating-systems evolution, we can draw an interesting analogy between operating systems and database systems. The concepts of the modern operating systems, programming languages, data structures, and shared resources are analogous to the concepts of modern database systems, data languages, data models and shared databases. Since a modern operating system executes and supports the user's programs in different programming languages and data structures, a modern database system should also execute and support the user's transactions in different data languages and data models. Since a modern operating system provides access to and management of a common set of resources for the running programs, a modern database system should also provide access to and management of a large collection of shared databases for the running transactions. Finally, as a modern operating system provides many modes of access, such as interactive programming and batch processing, a modern database system should also provide many modes of access, such as ad-hoc queries and transaction processing. With this analogy, we say that the multiple-backend database system should support multiple data models and their different data languages and provide various modes of access to the databases. Such a modern database-system capability is termed the multi-lingual database-system (MLDS) capability [16].

The MLDS capability has been demonstrated in MBDS. At the present, five different data models and their data languages are being supported by MBDS. They are the relational model and SQL, the hierarchical model and DL/1, the Codasyl model and DML, the entity-relationship model and Daplex, and the attribute-based model and ABDL. Basically, for the MBDS, the database is constructed in the attribute-based model, and the primary operations are a realization of the ABDL. All databases structured under other data models are converted into the attribute-based storage structures. All transactions written in other data languages such as SQL, DL/1, DML or Duplex are translated into the ABDL transactions [17-20]. In this way, there is only one database system with many and expandable translators and converters. Since all databases have the same storage structures, it is possible to share databases among transactions. For example, a SQL transaction accesses a hierarchical database or even a Codasyl database.

3.5. On Providing the Very Large Database Performance

As a parallel computer, the multiple-backend databases computer can conduct parallel processing and accessing with ease. The performance gain in terms of response-time reductions or in numbers of transaction executions per time unit (i.e., throughput) are achieved through a high degree of parallel processing and accessing. In other words, we are essentially configuring the multiple-backend computer for multiple-instructions-and-multiple-data-streams (MIMD) operations. For the multiple-backend database computer, these instructions are, of course, meta-and-base-data-access-and-processing operations; and the data streams consist of index pages and record blocks. Consequently, in order to maintain a high degree of MIMD operations, the database must be stored in the disk system in a multiple-data-streams fashion.

At the database-creation time, the meta (base) data of a database must be partitioned (clustered). Each partition (cluster) must be placed on the respective disks of the separate backends one page (block) at a time. For a round-robin database placement algorithm, if a partition (cluster) is, for example, of 25 pages (blocks) and the first available disk track to be used is at Backend 2, then for a 10-backend database system Backend 2 through Backend 6 will have 3 pages (blocks) of indices (records) on each of their respective disks, while Backend 1 and Backend 7 through Backend 10 will have only two pages (blocks) or indices (records) on each of their respective disks. (See Figure 6.)

The controller is responsible for determining the first page (block) (i.e., the first backend) to be used for the data placement and the backends are responsible for placing the indices (records) on their available tracks. Although different partitioning (clustering) schemes and database placement algorithms may be utilized for a given system, the design of the schemes and algorithms is to create the partition(cluster)-parallel-and-page(block)-serial effect for the subsequent access operations of the system. More specifically, in the above example we can conclude intuitively that the access and process times for the 25 pages (blocks) of indices (records) are shortened to the times for 2 or 3 pages (blocks) of indices (records). Thus, a 10-backend database computer may have a throughput of, at least, 8 times that of a single-backend database computer or of a mainframe-based database system.

As new indices (records) are being inserted into a database, the database placement algorithm will be activated frequently to place the new records on the next available pages (blocks). This does not require any redistribution of the database. However, as the new backends are being added to the system, it becomes necessary to execute the database placement algorithm for the entire existing database in order to maintain the optimal effect of partition(cluster)-parallel-and-page(record)-serial operations. This is termed the redistribution of the database. Such redistribution, although time-consuming, is infrequent (i.e., new backends are not added every day), has the desirable effect on system performance (i.e., new distribution of partitions and clusters allows a higher degree of parallel access and processing operations), and can be performed during the off-hours.

The concurrent execution of the index-partition operations of one transaction and the record-cluster operations of another transaction is facilitated by maintaining two

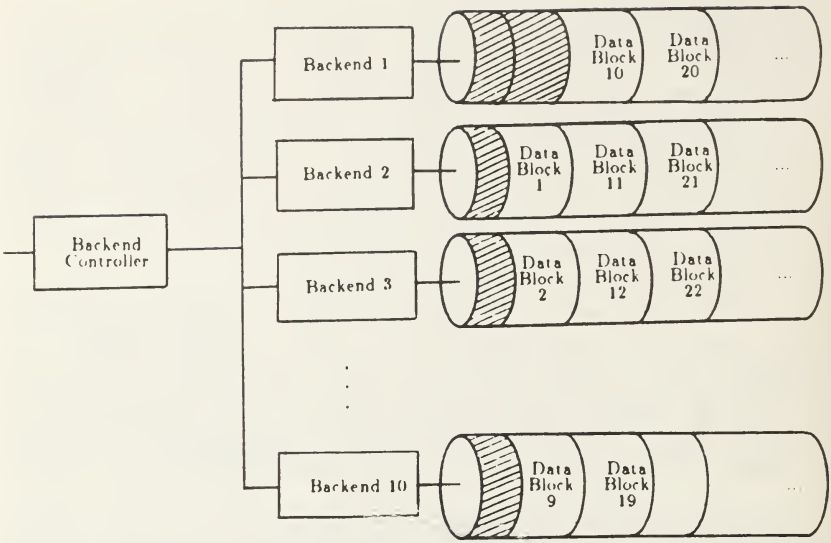

Figure 6. A Round-robin Database Placement Algorithm for Placing 25 Blocks of Base Data in a 10-Backend Database Computer.

queues at each backend, one for the index-partition operations and the other for the record-cluster operations. The concurrency control mechanisms of the backends make certain such concurrent executions of operations from the different queues do not violate the requirement of sequentiality for a given transaction.

In addition to parallelism and concurrency at the level of database operations, there is a way to improve the performance of the multiple-backend database computer at the architectural level. This way is similar to the way we have employed to accommodate response and database growth. In other words, we try to make performance gains in terms of the response-time reductions. For example, in a certain configuration X, we have x number of backends and one distribution of the database. In configuration Y, we have y number of backends and a redistribution of the same database. Since all of the software and the hardware of the backend, the backend controller and the interconnecting network are the same, the formula in Figure 7 establishes the performance-gains measure of configuration Y with respect to configuration X for a specific transaction and database.

$$\text{The Response-Time Reduction} = 1 - \left(\frac{\text{The Response Time of Configuration } Y}{\text{The Response Time of Configuration } X} \right)$$

Figure 7. The Response-Time-Reduction Formula

Again, let x, the number of backends for configuration X, be 20, and y, the number of backends for configuration Y, be 60, then ideally we would like the ratio of response times of configuration Y of 60 backends and configuration X of 20 backends to be 1/3. Consequently, the response-time reduction of the 60-backend database computer over the 20-backend database computer would be 2/3. This example illustrates that if we triple the number of backends of an existing multiple-backend database computer and redistribute the same database on the existing and new disks, we would expect to cut the response time of a transaction by two-thirds. In other words, the response-time reduction is inversely proportional to the ratio of the numbers of backends of the two configurations.

In reality, the response-time reductions are not likely to reach their ideal proportions. The issue is therefore how close a given multiple-backend database computer can reach its ideal response-time reductions, although preliminary benchmarking results are excellent for a small number of backends. As shown in Figure 8, we must measure (benchmark) a sufficient number of configurations for a given transaction and database in order to determine the system overheads and their impact on the response times (therefore, on the response-time reductions) of the transaction and the database.

Ideally, we would expect that in Figure 8, $R_i = \frac{1}{i}$ for $i = 1, 2, \ldots, n$, and for large n. Typically, $\Delta_1 = 0$ and $\Delta_2 < \Delta_3 < \cdots < \Delta_n$ where Δ_i is the system overhead incurred in handling the transaction in the i-backend configuration. In studying Figure 8, we may expect the benchmarking effort to address the following issues:

(1) What are the values of Δ_i for the given i-backend computers under benchmarking?

(2) How large will n be when there is no further reduction in response time (i.e., $R_n \geq R_{n+1}$)?

(3) How large will n be when the system overhead becomes pronounced (i.e., $\frac{\Delta_{n+1}}{n+1} \gg \frac{\Delta_n}{n}$)?

Even if the overheads (i.e., Δ_i) are not very low and the threshold (i.e., n where $R_n \geq R_{n+1}$) is not very large, this approach to performance gains (i.e., the response-time reduction) is attractive. We do not have to restore to the new and expensive hardware replacement and the lastest software version, we simply add same backends and their disk systems, replicate the system software and redistribute the database. This is a more cost-effective and non-disruptive upgrade of the database computer for performance gains [21].

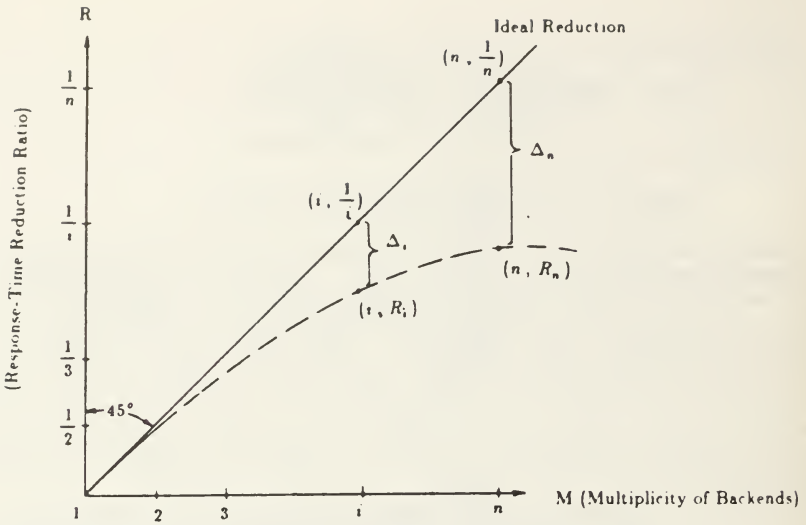

Figure 8. The Response-Time Reduction Measure

3.6. On Minimizing the Very Large Database Costs

Let us consider the cost issue in four different categories: upgrade cost, system cost, operational cost and application cost. The upgrade cost is associated the cost of upgrading the multiple-backend database computer for either performance gains or capacity growth. In either case, the emphasis of the upgrade is on expanding the multiple-backend database computer with more and same backend hardware and software. Such upgrade is definitively more cost-effective and less disruptive than the conventional approach of relying on an advanced hardware model and new software version. In addition, the architecture of the multiple-backend database computer tends to correlate the number of additions directly to the degree of performance gains. In other words, it pays to double or triple the number of backends, since the response-time reductions and the database-capacity increases tend to double or triple too. Neither the mainframe-based nor the single-backend database computer can be double or tripled with replicated software and identical hardware for upgrade.

On the operational side, the cost of database redistribution is the only additional one which is not found in the mainframe-based and single-backend DBMS. However, the redistribution operation is only needed at the time when the new backends and their disks are added. Consequently, it is not frequently done. As an automatic operation performed by the database placement system software, it can use the off-hours for redistribution. The cost is therefore minimal. The payoff in terms of MIMD operations is high.

On the application side, the cost of multi-model and multi-lingual software is not found in the conventional DBMS. In order to have the same capability of multi-model and multi-lingual, there must be multiple DBMSes running on several mainframe-based or single-backend computers. Obviously, multiple, separate DBMSes on their respective computers cost more than a single multiple-backend database system with an integrated user-system interface for multi-lingual translation and multi-model conversion. Furthermore, the former does not allow database sharing and the latter encourages database exchange. Finally, the cost and effort of upgrading several DBMSes and their mainframes or single backends is higher than the cost and effort of upgrade of a single multiple-backend database computer.

On system cost, the use of a backend controller and an interconnecting network seems to be a higher initial investment. However, this investment allows us to have a variably parallel system. As a parallel system, certain high-performance operations can be achieved cost-effectively. For example, since access and processing operations for indices (or records) are done in parallel, the effective access and processing rate of the indices (records) is several times higher than the physical rate of access, transfer and processing. Thus, we do not have to rely on expensive hardware for improving the physical rates of access, transfer and processing. Instead, we simply utilize a large number of standard hardware and software for the effective rates.

3.7. Concluding Remarks

The multiple-backend database computer may indeed have the hardware and software solutions to the efficient processing of very large databases. Its use of distributed management and control of disk systems enables the database computer to handle very large numbers of disk drives and therefore very large databases. Its variably parallel architecture permits the addition of identical hardware and replicated software to compensate for the growth of the database and the increase in the size of responses to the transactions. This compensation holds the response times of the transactions constant. By providing the multi-model conversion and multi-lingual translation, a wide range of databases and large number of data languages become available in the multiple-backend computer. The transaction for diverse and complex applications can be written and databases for different models can be shared. The issue of complexity is no longer present.

The multiple-backend database computer also provides software solutions such as the data placement and redistribution algorithms based on the notions of index partitions, record clusters, index-serial-and-partition-parallel and record-serial-and-cluster-parallel processing. By providing a parallel architecture, the multiple-backend database computer is carrying out the index-serial-and-partition-parallel and the record-serial-and-cluster-parallel operations in the MIMD fashion. Consequently, the performance should remain high.

The multiple-backend database computer is cost-effective. By relying on off-the-shelf hardware, replicable software and built-in expandability, the multiple-backend database computer does not have to rely on the traditional and conventional approach to upgrade, and thereby achieves more orderly, inexpensive and flexible upgrades.

As a new architectural solution to very large database management and processing, the multiple-backend database computer will be subject to more studies, benchmarks and improvements. In this paper, we hope that we have at least provided an introduction to its arrivals at the scene of very large database management and processing.

ACKNOWLEDGEMENT

The author would like to thank Professor Dr. Antonia L. Furtado of Pontificia Universidade Catolica do Rio de Janeiro for the suggestion of the title of this paper. As a member of the Programme Committee of IFIP Congress '86, Professor Dr. Furtado is instrumental in including this paper as an invited paper of the Congress. Without the invitation, this paper would have never been written. The author would also like to thank Professor Dr. Georges Gardarin of University of Paris VI and INRIA who as the responder of the paper has contributed the improvement of the paper. Finally, the author would like to thank Professor Dr. Thomas Wu of the Naval Postgraduate School and Mr. Steven A. Demurjian, a doctoral candidate of the Ohio State University, for reading and commenting a earlier draft of the paper.

REFERENCES

1. Steven, L. D., "The Evolution of Magnetic Storage," *IBM Journal of Research and Development,* Vol. 25, No. 5, (Sept. 1981).
2. Feudo, C. "Modern Hardware Technologies and Software Techniques for Online Database Storage and Access," Master's Thesis, Naval Postgraduate School, Monterey, CA 93943, (Dec. 1985).
3. Tsichritzis, D. C., and Lochovshy, F. H., *Data Models,* Prentice-Hall, Inc., (1982).
4. Cardenas, A. F., *Database Management Systems,* (Second Edition), Allyn and Bacon, Inc., (1985).
5. Codd, E. F., "A Relational Model of Data for Large Shared Data Banks," *Communications of ACM,* Vol. 13, No. 6, (June 1970).
6. Chamberlin, D. D., et al, "SEQUEL 2: A Unified Approach to Data Definition, Manipulation, and Control," *IBM Journal of Research and Development,* Vol. 20, No. 6 (Nov. 1976).
7. Tsichritzis, D. C., and Lochovsky, F. H. "Hierarchical Data Base Management: A Survey." *ACM Computing Survey,* Vol. 8, No. 1, (March 1976).
8. Kapp, D. and Leben, J. F., *IMS Programming Techniques: A Guide to using DL/I,* Van Nostrand Reihold, (1978).
9. "SQL/Data System, Concepts and Facilities," IBM Corp. Reference Manual GH24-5017.
10. "Information Management System/Virtual Storage," IBM Corp. General Information Manual GH20-1260.
11. "IDM Software Reference Manual, Version 1.4," Britton Lee, Inc., January 1983.

12. Canaday, R. W., et al, "A Back-End Computer for Data Base Management," *Communication of ACM,* Vol. 17, No. 10, (Oct. 1974).

13. Hsiao, D. K., et al, "The Implementation of a Multibackend Database Systems," Chapters 10 and 11, *Advanced Database Machine Architecture,* (Ed. D. K. Hsiao) Prentice-Hall, Inc., 1983.

14. "DBC/1012 Data Base Computer Concepts and Facilities," C02:001:00, Release 1.0, (April 1983).

15. Demurjian, S. A., Hsiao, D. K., and Menon, J., "A Multi-Backend Database System for Performance Gains, Capacity Growth and Hardware Upgrade," *Proceedings of Second International Conference on Data Engineering,* IEEE Computer Society, (Feb. 1986).

16. Demurjian, S. A., and Hsiao, D. K., "New Directions in Database-Systems Research and Development," *Proceedings of International Conference on New Directions in Computing,* IEEE Computer Society, (August 1985).

17. Kloepping, G. R., and Mack, J. F., "The Design and Implementation of a Relational Interface for the Multilingual Database System," Master's theis, Naval Postgraduate School, Monterey, CA, (June 1985).

18. Benson, T. P., and Wentz, G. L., "The Design and Implementation of a Hierarchical Interface for the Multilingual Database System," Master's thesis, Naval Postgraduate School, Monterey, CA, (June 1985).

19. Emdi, B., and Wortherly, R., "The Design and Implementation of a Network (Codasyl) Interface for the Multilingual Database System," Master's thesis, Naval Postgraduate School, Monterey, CA, (Dec. 1985).

20. Billings, J., and Antony, J. A., "The Design and Implementation of an Entity-Relationship (Daplex) Interface for the Multilingual Database System, Master's thesis, Naval Postgraduate School, Monterey, CA, (Dec. 1985).

21. Demjurjian, S. A., et al., "Performance Evaluation of a Database System in Multiple Backend Configurations," *Proceedings of the Fourth International Workshop on Database Machines,* MCC, (March 1985).

INITIAL DISTRIBUTION LIST

Defense Technical Information Center Cameron Station Alexandria, VA 22314	2
Dudley Knox Library Code 0142 Naval Postgraduate School Monterey, CA 93943-5100	2
Office of Research Administration Code 012A Naval Postgraduate School Monterey, CA 93943-5100	1
Chairman, Code 52V1 Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	40
David K. Hsiao Code 52Hq Computer Science Department Naval Postgraduate School Monterey, CA 93943-5100	150
Chief of Naval Research Arlington, VA 22217	1